

Lmy

***** Information Materials for IDS *****

To YOR IP Law Docket Number JP920030021 Date Sep/15/06

Prepared by Naoko Uekusa Date of JPO Office Action Sep/05/06

Applied Art (The following reference(s) were cited by JPO Examiner as Prior art to the following JP claims)

Ref.	Patent Document No. or Title	Publication Date (MM/DD/YY)	English abs. or counterpart document available (Y/N)	JP claim(s)
A				
B				
C				
D				
E				
F				

<NOTE>

- ◇ PUPA : Published Unexamined Patent Application ◇ PEPA : Published Examined Patent Application
 ◇ PUUMA : Published Unexamined Utility Model Application ◇ PEUMA : Published Examined Utility Model Application
 ◇ JP : Japanese Patent ◇ * : Reference being filed before and published after the priority application date of the subject docket

to be continued ☐

Background Art (The following reference(s) were cited but not applied to the JP claims.)

Ref.	Patent Document No. or Title	Ref.	Patent Document No. or Title
bgA	PUPA 05-197565	bgD	
bgB		bgE	
bgC		bgF	

Comment, if any :

to be continued ☐

006 OCT -3 AM 10:19
 RECEIVED
 JPO DEPT.

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 05-197565

(43)Date of publication of application : 06.08.1993

(51)Int.Cl.

G06F 9/45

(21)Application number : 04-008166

(71)Applicant : FUJITSU LTD

(22)Date of filing : 21.01.1992

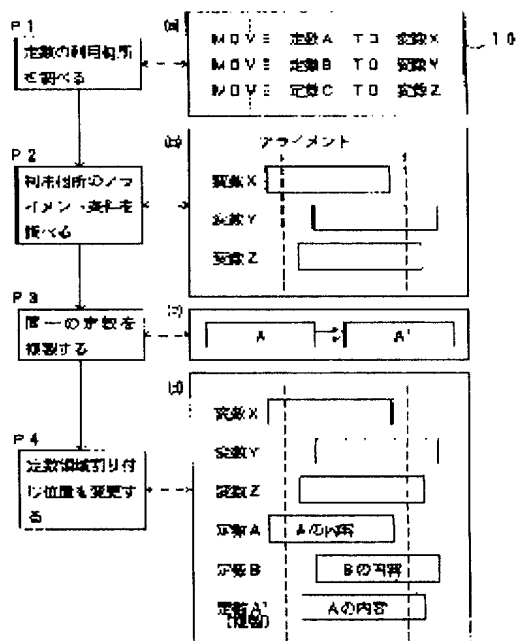
(72)Inventor : TAKAHASHI YOSHIO

(54) CONSTANT AREA ALLOCATION PROCESSING METHOD

(57)Abstract:

PURPOSE: To reduce the number of times referring to and updating main storage at the time of executing a processing using a constant, to speed up obtaining a translated result on a constant area allocation processing method in a compiler translating a source program into an object program.

CONSTITUTION: A place using the constant in the program is extracted by referring to a program information (P1), and an alignment condition at the place using the constant is checked (P2). When the same constant is used at plural places in the different alignment condition, the constant is copied by the number of the plural places (P3). Then, the allocation position of the constant is altered in accordance with the alignment condition of a destination using the constant (P4). Thus, optimization for adjusting the allocation areas of the respective constants to the alignment condition of a computer to be a target and reducing the number of instructions required for the reference of the respective constants is executed.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

(19)日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11)特許出願公開番号

特開平5-197565

(43)公開日 平成5年(1993)8月6日

(51)Int.Cl.⁵

G 0 6 F 9/45

識別記号

庁内整理番号

9292-5B

F I

G 0 6 F 9/ 44

技術表示箇所

3 2 2 H

審査請求 未請求 請求項の数1(全 10 頁)

(21)出願番号 特願平4-8166

(22)出願日 平成4年(1992)1月21日

(71)出願人 000005223

富士通株式会社

神奈川県川崎市中原区上小田中1015番地

(72)発明者 高橋 義雄

神奈川県川崎市中原区上小田中1015番地

富士通株式会社内

(74)代理人 弁理士 小笠原 吉義 (外2名)

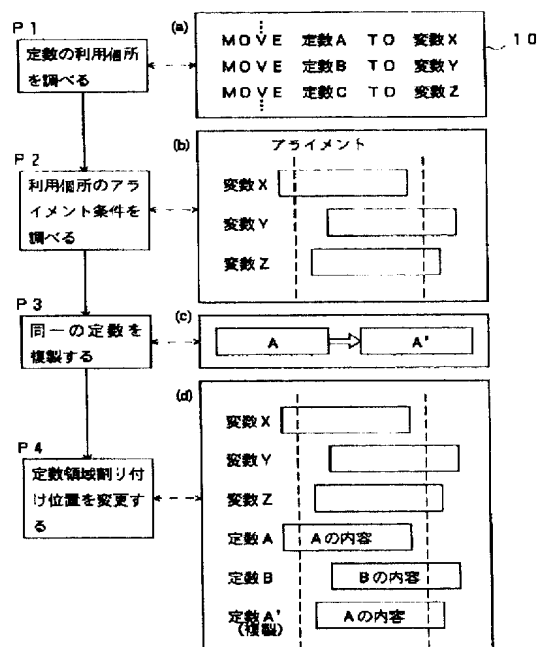
(54)【発明の名称】 定数領域割り付け処理方法

(57)【要約】

【目的】 ソースプログラムをオブジェクトプログラムに翻訳するコンパイラにおける定数領域割り付け処理方法に関し、定数を利用した処理を行う際の主記憶参照および更新の回数を削減し、翻訳結果の高速化を実現することを目的とする。

【構成】 プログラム情報を参照して、プログラム中の定数を利用する個所を抽出し(P1)、定数を利用する個所におけるアライメント条件を調べる(P2)。同一の定数を異なるアライメント条件のもとに複数個所で利用していれば、その定数をその複数個所分だけ複製する(P3)。そして、定数の割り付け位置を、定数を利用する先のアライメント条件に従って変更する(P4)。こうして、各定数の割り付け領域をターゲットとなる計算機のアライメント条件に合わせ、各定数の参照に必要な命令数を少なくする最適化を行う。

本発明の原理説明図



【特許請求の範囲】

【請求項1】 ソースプログラムをオブジェクトプログラムに翻訳するコンパイラにおける定数領域割り付け処理方法において、プログラム情報を参照して、プログラム中の定数を利用する個所を抽出する過程(P1)と、定数を利用する個所におけるアライメント条件を調べる過程(P2)と、同一の定数を異なるアライメント条件のもとに複数個所で利用しているとき、その定数をその複数個所分だけ複製する過程(P3)と、定数の割り付け位置を、定数を利用する先のアライメント条件に従って変更する過程(P4)とを備え、各定数の割り付け領域を、ターゲットとなる計算機によって決定されるアライメント条件に合わせ、各定数の参照に必要な命令数を少なくする最適化を行うことを特徴とする定数領域割り付け処理方法。

【発明の詳細な説明】

【0001】

【産業上の利用分野】 本発明は、アライメント条件に制限がある計算機で動作するプログラムのコンパイル処理方法に係り、特に、定数をその利用先のアライメントに従って割り付けることにより、定数参照の高速化を図ったコンパイラにおける定数領域割り付け処理方法に関する。

【0002】 計算機の中央処理装置(CPU)の高速化に伴い、CPUの実行速度と主記憶への参照および更新速度との格差が問題となっている。このため、各種アーキテクチャの計算機において、効率のよい主記憶の参照と更新(以下、メモリアクセスという)方法が必要とされている。

【0003】

【従来の技術】 一般に、コンパイラは、ソースプログラムのテキストの字句解析および意味解析を行って、定数として扱うことができるデータを集め、それらの定数の領域を割り付ける処理を行っている。

【0004】 図5は従来の定数領域割り付けの例を示す図、図6は従来の命令の生成例説明図である。図5の(イ)に示すソースプログラム10は、COBOL言語で記述されたプログラムであり、この例では、2つの文字列定数"first", "second"を、変数Xおよび変数Yへ転記している。このソースプログラム10の翻訳処理では、変数領域の先頭が8バイトアライメントにあり、図5の(ロ)に示すように、変数Xの領域が0から9のアドレス、変数Yの領域が10から19のアドレスに割り付けられたとする。なお、8バイトアライメントとは、アドレス値が8で割り切れる値になっていることを意味する。

【0005】 従来のコンパイラにおいては、文字列定数は、例えば図5の(ハ)に示すように、連続した領域に順番に割り付けられる。この例では、文字列定数"first"が定数領域アドレスの128から132まで、文字列定数"second"が定数領域アドレスの13

3から138まで割り付けられている。

【0006】 ところで、最近のハードウェアには、例えば2バイト単位のメモリアクセスは2バイト単位の開始アドレスからしか行えず、4バイト単位のメモリアクセスは4バイト単位の開始アドレスからしか行えないというように、メモリアクセスの単位量とメモリアクセスの開始位置とを任意に選ぶことができないアーキテクチャをとるものがある。

【0007】 このようなアライメント条件に制限があるアーキテクチャの場合には、文字列定数"second"を転記する操作などの際に、開始アドレスが合わないために、複数バイトずつまとめて転記できないことがある。

【0008】 図6の(イ)は、図5の(イ)に示すソースプログラム10中の2つのMOVEステートメントを翻訳した例を示している。同図では、説明を分かりやすくするために、翻訳結果をアセンブラ命令の形式で表している。ここで、各命令は次の機能を持っている。

【0009】 load-address: 第1オペランドのアドレスを、第2オペランドにロードする。

load-4bytes: 第1オペランドの内容(4バイト)を、第2オペランドにロードする。

【0010】 store-4bytes: 第1オペランドの内容(4バイト)を、第2オペランドの示すアドレスにストアする。

load-1byte: 第1オペランドの内容(1バイト)を、第2オペランドにロードする。

【0011】 store-1byte: 第1オペランドの内容(1バイト)を、第2オペランドの示すアドレスにストアする。

このプログラムは、まず変数Xのアドレスをレジスタr0にロードし、次に図5の(ハ)に示す定数領域の128番地から4バイトの文字列定数"first"をレジスタr1にロードし、それをレジスタr0の示すアドレスにストアしている。続いて、定数領域の132番地から1バイトの文字列定数"t"をレジスタr1にロードし、それをレジスタr0の示すアドレスから4バイト目のアドレスにストアし、以下、変数Yのアドレスをレジスタr0にロードし、同様に"s","e","c","o","n","d"を1バイトずつロードして、変数Yの領域に順番にストアしている。

【0012】 この結果、文字列定数"first", "second"の変数Xおよび変数Yへの転記は、図6の(ロ)に示すように行われることになる。ここで、文字列定数"second"が1バイトずつしか転記されないのは、定数"second"が奇数番地から始まり、変数Yが偶数番地から始まるため、2バイト単位以上まとめてロード/ストアすることができないからである。

【0013】

【発明が解決しようとする課題】 以上のように、従来の

コンパイラにおいては、定数を参照するために必要な命令数に関する考慮を払うことなく、集めた定数を機械的に連続した領域に割り付けているため、転記元と転記先の開始アドレスが合わないときに、複数バイトをまとめて転記することができず、命令数が多くなって、命令の実行時間が長くなるという問題があった。

【0014】本発明は上記問題点の解決を図り、定数を利用した処理を行う際の主記憶参照および更新を効率よく行うことができるようにし、翻訳したプログラムの高速実行を可能とすることを目的としている。

【0015】

【課題を解決するための手段】図1は本発明の原理説明図である。ソースプログラムをオブジェクトプログラムに翻訳するコンパイラにおいて定数領域を割り付ける際に、字句・意味解析で見つけた定数を、単純に連続領域に割り付けるのではなく、それぞれの定数が利用される先のアライメント条件を調べ、翻訳結果を動作させるターゲットとなる計算機上でのメモリアクセスの効率が最大限になるように、定数割り付け領域の最適化を行う。

【0016】そのため、処理過程P1では、字句・意味解析を行った結果のプログラム情報を参照して、プログラム中において定数を利用する個所を抽出する。処理過程P2では、処理過程P1で抽出した定数の利用個所におけるアライメント条件を調べる。すなわち、定数を転記する個所が何バイト単位から始まっているかなどを調べる。

【0017】処理過程P3では、同一の定数を異なるアライメント条件のもとに複数個所で利用しているとき、その定数をその複数個所分だけ複製する。処理過程P4では、定数の割り付けを位置を、定数を利用する先のアライメント条件に従って変更する。例えば、定数の利用先が4バイトアライメントになっていれば、その定数についても4バイトアライメントに合わせるように割り付ける。

【0018】これにより、各定数の割り付け領域を利用先のアライメントに合わせ、定数参照に必要な命令数を少なくする。

【0019】

【作用】本発明は、ハードウェアアーキテクチャにより決定されるアライメント条件を守りながら、定数をその利用先のアライメントに従って割り付けることにより、定数参照の高速化を図るものである。

【0020】処理過程P1では、例えば図1の(a)に示すようなソースプログラム10の定数の利用個所を調べる。この例では、変数X、Y、Zにそれぞれ定数A、B、Aを転記している。処理過程P2では、定数利用個所のアライメント条件、すなわち、変数X、Y、Zが各々どのようなアライメントで配置されているかを調べる。処理過程P3では、同一の定数Aを利用する個所が2箇所あるので、定数Aの複製A'を作る。処理過程P

4では、定数A、定数B、定数A'のアライメントを、それぞれ変数X、Y、Zのアライメントに合わせるように、各定数領域の割り付け位置を変更する。

【0021】こうすることにより、定数A、B、A'を変数X、Y、Zに転記する際に、複数バイトずつまとめて転記する命令を用いることができるようになる。

【0022】

【実施例】図2は本発明の実施例による処理構成を示すブロック図、図3は本発明の実施例による定数領域割り付けの例を示す図、図4は本発明の実施例による命令の生成例説明図である。

【0023】図2において、ソースプログラム10は、COBOL言語などの高級言語で記述された翻訳対象のプログラムである。処理装置11は、CPUおよびメモリなどからなるコンパイル処理装置である。コンパイラ12は、ソースプログラム10を機械語レベルのオブジェクトプログラム22に翻訳するプログラムである。

【0024】ソース解析部13は、ソースプログラム10の字句解析および意味解析を行い、解析結果を中間コードなどのプログラム情報14として出力する。最適化処理部15は、プログラム情報14を参照して、処理の最適化を行うものである。本発明は、特にこの最適化処理部15の処理に関係している。

【0025】アーキテクチャ情報16は、ターゲットとなる計算機のアライメント条件などに関する事前に与えられる情報である。アライメント条件判定処理部17は、アーキテクチャ情報16を参照し、オブジェクトプログラム22を動作させる計算機に、アライメント条件に関する制限があるかどうか、すなわちメモリアクセスの単位量とメモリアクセスの開始位置との間に、何らかの制限があるかどうかを判定する。制限がない場合、従来と同じ定数領域の割り付けを行う。

【0026】アライメントに関する制限が存在する場合、利用先確認処理部18は、プログラム情報14を参照して、定数の利用先を調べる。また、同一の定数がプログラム中の複数個所で参照されているかどうかを調べる。

【0027】同一の定数が複数個所から参照されている場合に、領域複製処理部19は、その定数を参照先の個数分だけ複製する。定数領域割り付け位置変更処理部20は、定数の利用先のアライメント情報をプログラム情報14から入手し、それに合わせて各定数の割り付け位置を変更する処理を行う。変更した位置に関する情報は、プログラム情報14として格納しておく。

【0028】オブジェクト生成処理部21は、プログラム情報14に従って、オブジェクトプログラム22を生成する処理を行うものである。以上により、ハードウェアのアライメント条件のもとで、高速に参照可能な定数領域の割り付けが実現されることになる。

【0029】例えば、図3の(イ)に示すようなCOB

20

30

40

50

OL言語で記述されたソースプログラム10をコンパイルするものとする。このソースプログラム10は、従来技術の説明で用いた例と同じである。

【0030】変数Xは、図3の(ロ)に示すように、8バイトアライメントにある変数領域アドレスの0から確保され、変数Yは、変数領域アドレスの10から確保されている。この変数Xおよび変数Yの確保されるアドレスは、TESTという名前で構造化されているため、変更することはできない。

【0031】変数X、変数Yに代入する文字列定数"first", "second"の領域を割り付ける場合、
10 変数Xは、変数領域アドレスの8バイトアライメントから確保されているので、文字列定数"first"も8バイトアライメントから確保する。この例では、定数領域アドレスの128に文字列定数"first"を割り付けている。

【0032】一方、変数Yは、変数領域アドレスの10、すなわち8バイトアライメントより2バイト後の位置に確保されているので、文字列定数"second"
20 は8バイトアライメントより2バイト後に確保する。そのため、例えば図3の(ハ)に示すように、文字列定数"first"の後に5バイトの未使用領域を設け、文字列定数"second"を定数領域のアドレス138から確保する。

【0033】図4の(イ)は、図3の(イ)に示すソースプログラム10中の2つのMOVEステートメントを翻訳した例を示している。ターゲットの計算機は、いわゆるRISCアーキテクチャの命令セットを採用する計算機で、メモリアクセスの単位量とメモリアクセス開始位置とを任意に選ぶことができないというアライメント
30 条件の制限がある。図4の(イ)に示す各命令は次の機能を持っている。

【0034】load-address：第1オペランドのアドレスを、第2オペランドにロードする。

load-1byte：第1オペランドの内容(1バイト)を、第2オペランドにロードする。

【0035】store-1byte：第1オペランドの内容(1バイト)を、第2オペランドの示すアドレスにストアする。

load-2bytes：第1オペランドの内容(2バイト)を、
40 第2オペランドにロードする。

【0036】store-2bytes：第1オペランドの内容(2バイト)を、第2オペランドの示すアドレスにストアする。

load-4bytes：第1オペランドの内容(4バイト)を、第2オペランドにロードする。

【0037】store-4bytes：第1オペランドの内容(4

バイト)を、第2オペランドの示すアドレスにストアする。

このプログラムは、まず変数Xのアドレスをレジスタr0にロードし、次に図3の(ハ)に示す定数領域の128番地から4バイトの文字列定数"first"をレジスタr1にロードし、それをレジスタr0の示すアドレスにストアする。続いて、定数領域の132番地から1バイトの文字列定数"t"をレジスタr1にロードし、それをレジスタr0の示すアドレスから4バイト目のアドレスにストアする。

【0038】次に変数Yのアドレスをレジスタr0にロードし、変数Yの開始位置と文字列定数"second"の開始位置が、ともに8バイトアライメント+2の位置にあるので、先頭の2バイト"se"をロードし、それをレジスタr0の示すアドレスにストアする。残りの文字列定数"cond"は8バイトアライメント+4、すなわち4バイトアライメントにあるので、4バイトをまとめてロードし、レジスタr0の内容+2のアドレス(変数領域のアドレス12)にストアする。

【0039】この例では、定数の参照箇所がそれぞれ1箇所であるが、もし同一内容の定数が複数箇所参照される場合には、アライメントを合わせるのに必要な分の定数の複製を作ることにより、定数領域割り付け位置のアライメントを、各利用先のアライメントに合わせる。

【0040】このように、アライメントを合わせて領域を確保するため、複数バイトをまとめてコピーすることができる。したがって、図6に示す従来方法と比べると明らかなように、定数参照の命令数を削減し、翻訳結果の高速化が実現される。

【0041】

【発明の効果】以上説明したように、本発明によれば、主記憶に対する参照および更新の回数を削減でき、翻訳結果の高速化が可能になる。したがって、計算機全体の性能向上に寄与するところが大きい。

【図面の簡単な説明】

【図1】本発明の原理説明図である。

【図2】本発明の実施例ブロック図である。

【図3】本発明の実施例による定数領域割り付けの例を示す図である。

【図4】本発明の実施例による命令の生成例説明図である。

【図5】従来の定数領域割り付けの例を示す図である。

【図6】従来の命令の生成例説明図である。

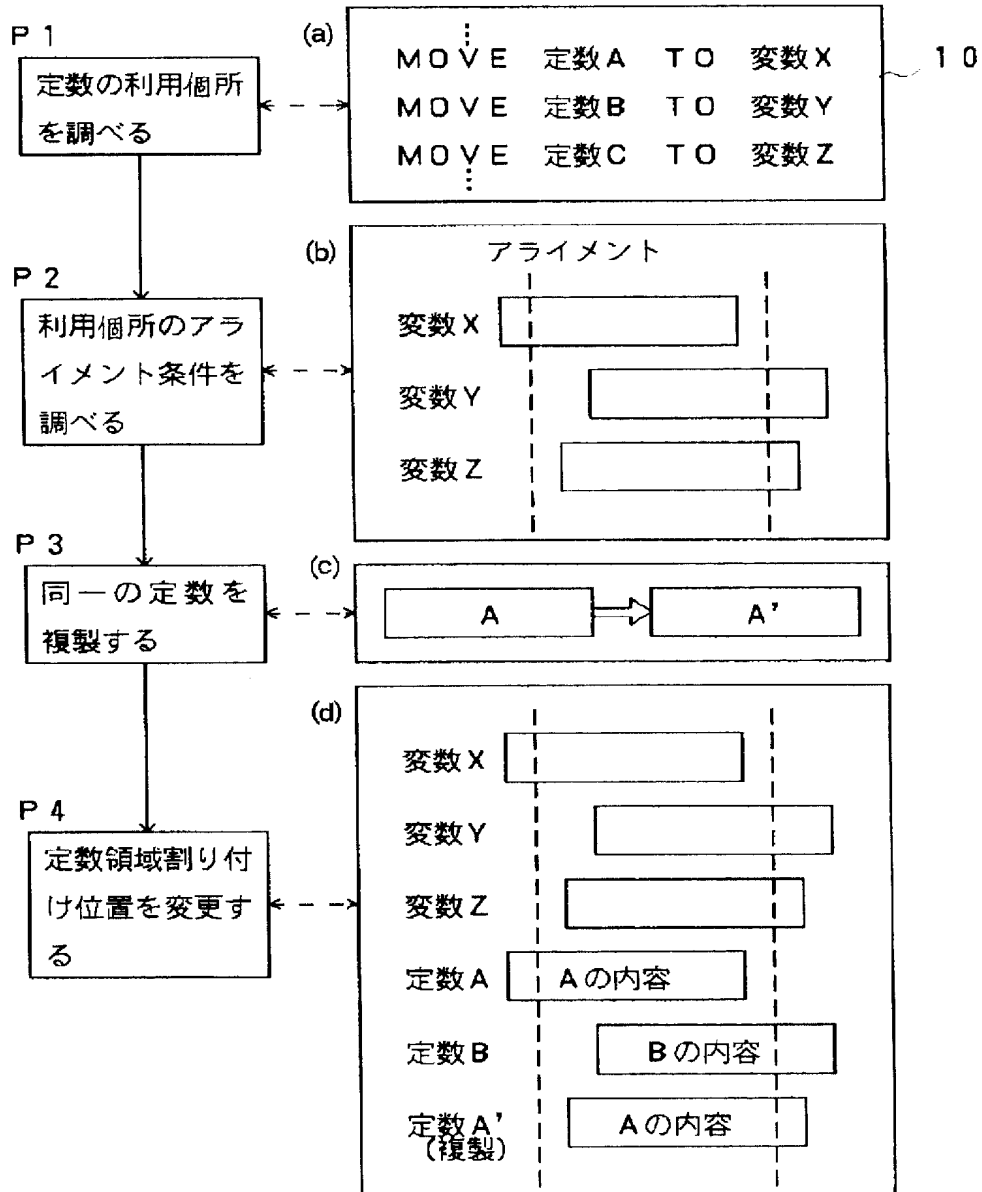
【符号の説明】

10 ソースプログラム

P1～P4 本発明の処理過程

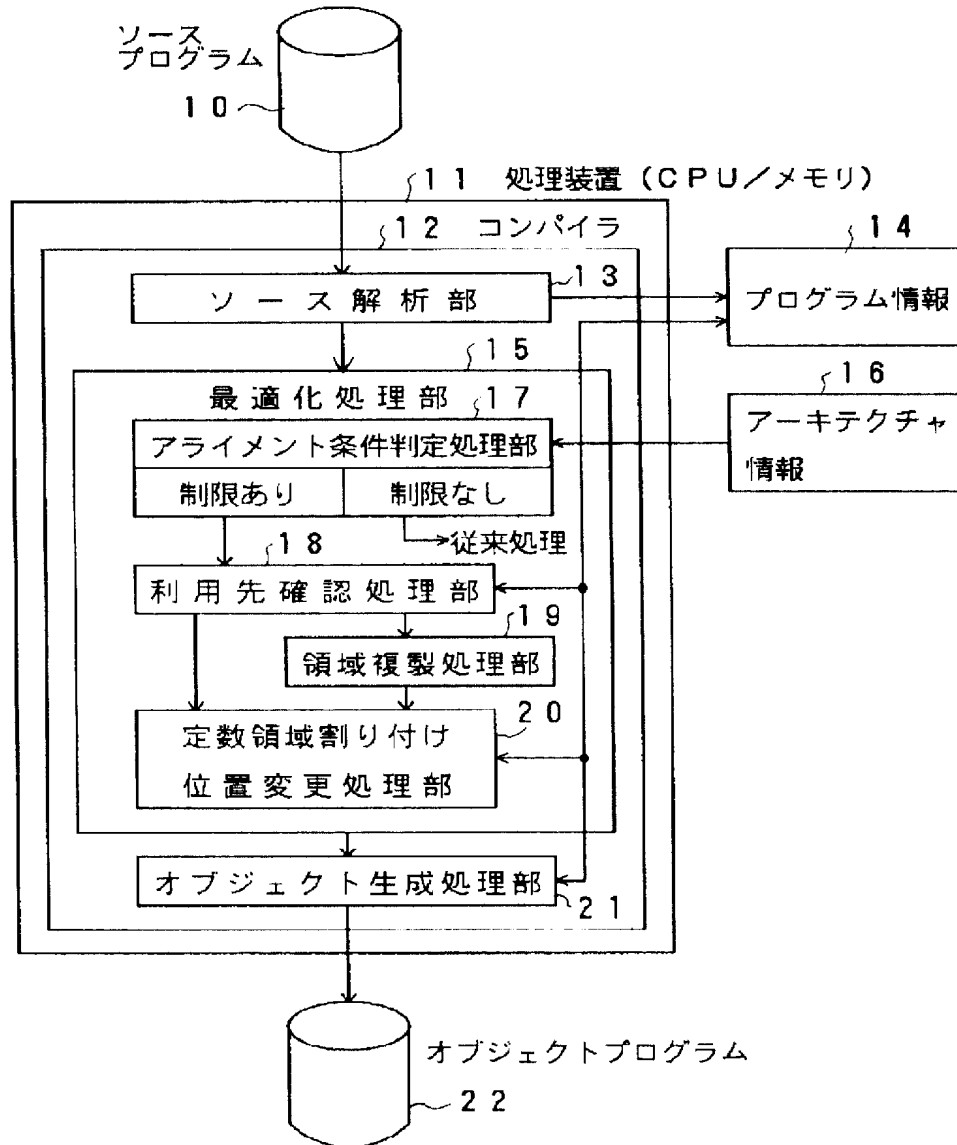
【図1】

本発明の原理説明図



【図2】

本発明の実施例ブロック図



【図3】

定数領域割り付けの例

(イ) ソースプログラム

```

01 TEST.
02 X PIC X(10).
02 Y PIC X(10).

...
MOVE "FIRST" TO X.
MOVE "SECOND" TO Y.

```

~ 10

(ロ) 変数領域

0 1 2 . . . 9 10 11 . . . 19

X の 領 域	Y の 領 域
---------	---------

(ハ) 定数領域

定数領域 アドレス	16進 表現	ASCII 表 現	8バイトアライメン トからのオフセット
128	66	f	+0
129	69	i	+1
130	72	r	+2
131	73	s	+3
132	74	t	+4
133	例では未使用		+5
134	例では未使用		+6
135	例では未使用		+7
136	例では未使用		+0
137	例では未使用		+1
138	73	s	+2
139	65	e	+3
140	63	c	+4
141	6F	o	+5
142	6E	n	+6
143	64	d	+7

【図4】

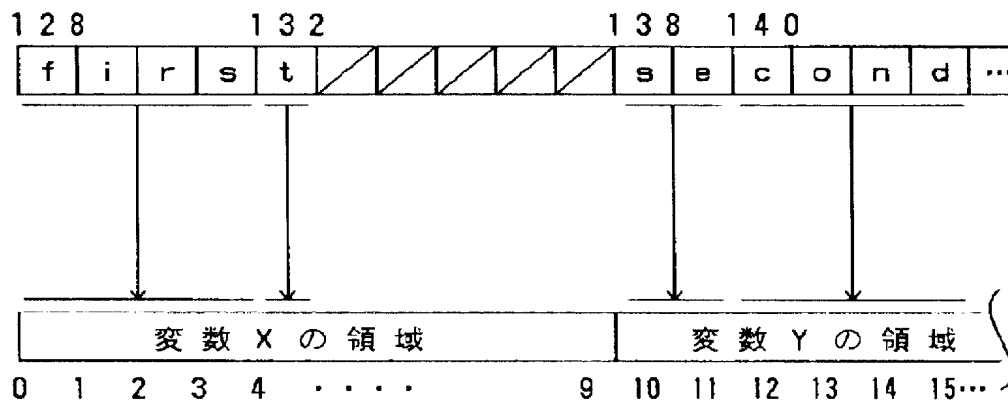
命令の生成例説明図

(イ)

load_address	x, r0	; 変数Xのアドレスロード
load_4bytes	128, r1	; "firs"のコピー
store_4bytes	r1, [r0]	
load_1byte	132, r1	; "t" のコピー
store_1byte	r1, [r0+4]	
load_address	y, r0	; 変数Yのアドレスロード
load_2bytes	138, r1	; "se"のコピー
store_2bytes	r1, [r0]	
load_4bytes	140, r1	; "cond"のコピー
store_4bytes	r1, [r0+2]	

(ロ)

定数領域



【図5】

従来の定数領域割り付けの例

(イ) ソースプログラム

```

01 TEST.
02 X PIC X(10).
02 Y PIC X(10).

...
MOVE "FIRST" TO X.
MOVE "SECOND" TO Y.

```

~ 10

(ロ) 変数領域

0	1	2	...	9	10	11	...	19
X の 領 域					Y の 領 域			

(ハ) 定数領域

定数領域 アドレス	16進 表現	ASCII 表 現	8バイトアライメン トからのオフセット
128	66	f	+0
129	69	i	+1
130	72	r	+2
131	73	s	+3
132	74	t	+4
133	73	s	+5
134	65	e	+6
135	63	c	+7
136	6F	o	+0
137	6E	n	+1
138	64	d	+2

【図6】

従来の命令の生成例説明図

(イ)

load_address	x, r0	; 変数Xのアドレスロード
load_4bytes	128, r1	; "firs"のコピー
store_4bytes	r1, [r0]	
load_1byte	132, r1	; "t" のコピー
store_1byte	r1, [r0+4]	
load_address	y, r0	; 変数Yのアドレスロード
load_1byte	133, r1	; "s" のコピー
store_1byte	r1, [r0]	
load_1byte	134, r1	; "e" のコピー
store_1byte	r1, [r0+1]	
load_1byte	135, r1	; "c" のコピー
store_1byte	r1, [r0+2]	
load_1byte	136, r1	; "o" のコピー
store_1byte	r1, [r0+3]	
load_1byte	137, r1	; "n" のコピー
store_1byte	r1, [r0+4]	
load_1byte	138, r1	; "d" のコピー
store_1byte	r1, [r0+5]	

(ロ)

定数領域

